RSAC 2025 Conference

Many Voices.
One Community.

SESSION ID: CLS-M03

Honey, I Shrunk the Accounts
Reducing Blast Radius Using AWS Small Accounts

Andy Ozment
Chief Technology Risk Officer, Capital One
04/28/2025

Disclaimer

Presentations are intended for educational purposes only and do not replace independent professional judgment. Statements of fact and opinions expressed are those of the presenters individually and, unless expressly stated to the contrary, are not the opinion or position of RSA Conference LLC or any other co-sponsors. RSA Conference LLC does not endorse or approve, and assumes no responsibility for, the content, accuracy or completeness of the information presented.

Attendees should note that sessions may be audio- or video-recorded and may be published in various media, including print, audio and video formats without further notice. The presentation template and any media capture are subject to copyright protection.

© 2025 RSA Conference LLC or its affiliates. The RSAC and RSAC CONFERENCE logos and other trademarks are proprietary. All rights reserved.

Andy Ozment

And a huge thank you to the teams who did the actual work in today's presentation:

Cloud Operations & Resilience Engineering

Enterprise Programs & Platforms

Cyber

Technology Risk Management

At Capital One, we're taking the approach that smaller is better, particularly with our AWS accounts

An AWS account is the foundation for consuming, managing, and protecting AWS resources

AWS accounts serve two primary functions:

Container for your resources

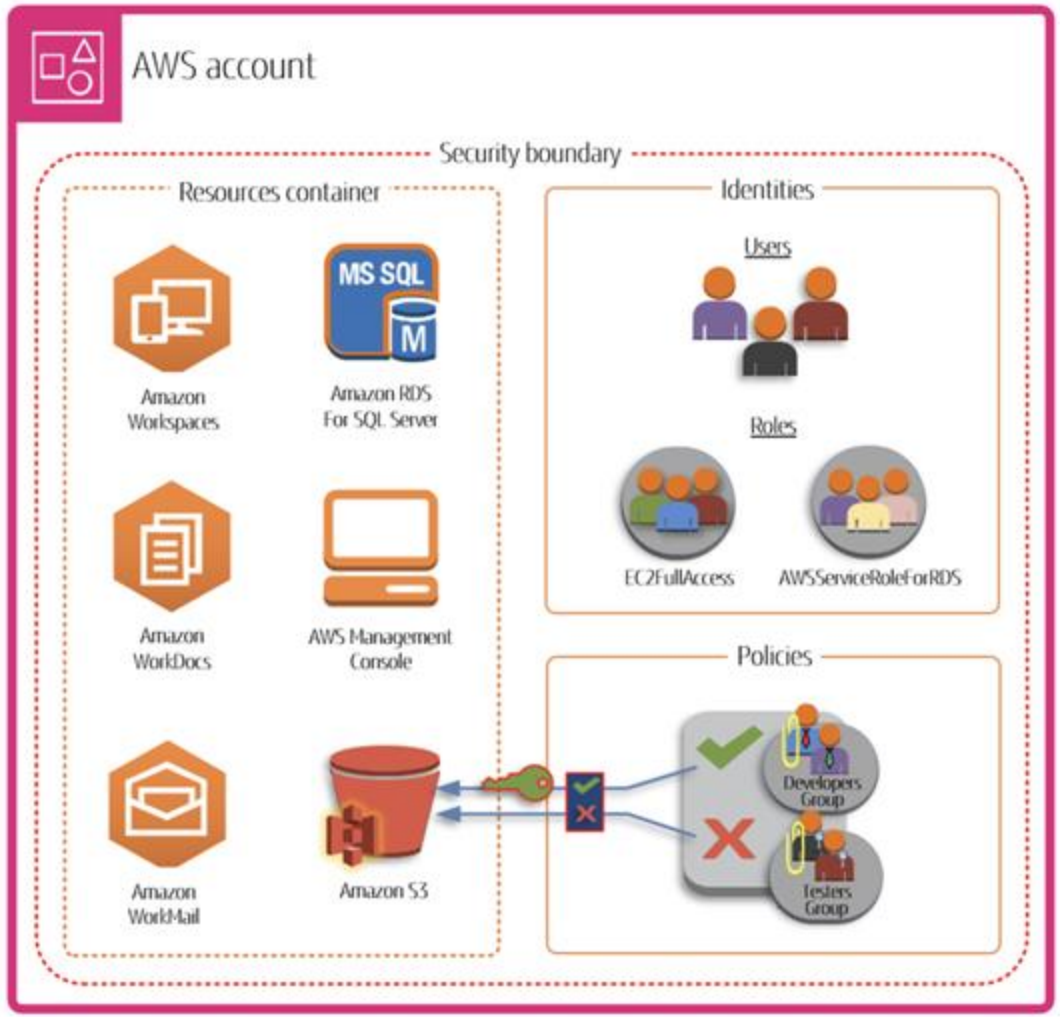
Security, access, and billing boundaries

AWS accounts provide the following features:

Logical resource isolation and segmentation

Group workloads into accounts

Monitor and control billing and costs



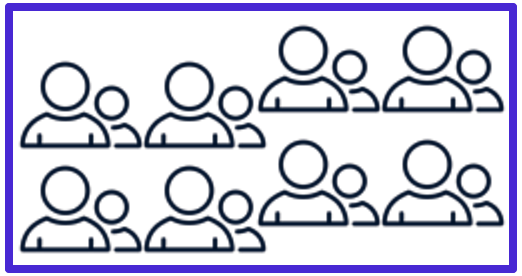
As AWS accounts grow larger, the complexity and risk also grow

Blast Radius

Operational Issues

Governance

Blast Radius: AWS accounts are the likely boundary for security incidents



Overly-permissive Identity and Access Management (IAM) is problematic, but limited to the account boundary

Co-mingling network and apps in one account increases risk of network configuration tampering

Operational Issues: Large accounts run the risk of resource exhaustion or hitting rate limits

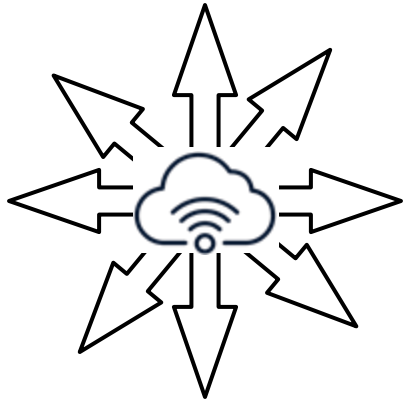
Application Programming Interface (API) rate limit exhaustion

Lambda concurrency limits

Classless Inter-Domain Routing (CIDR) block exhaustion

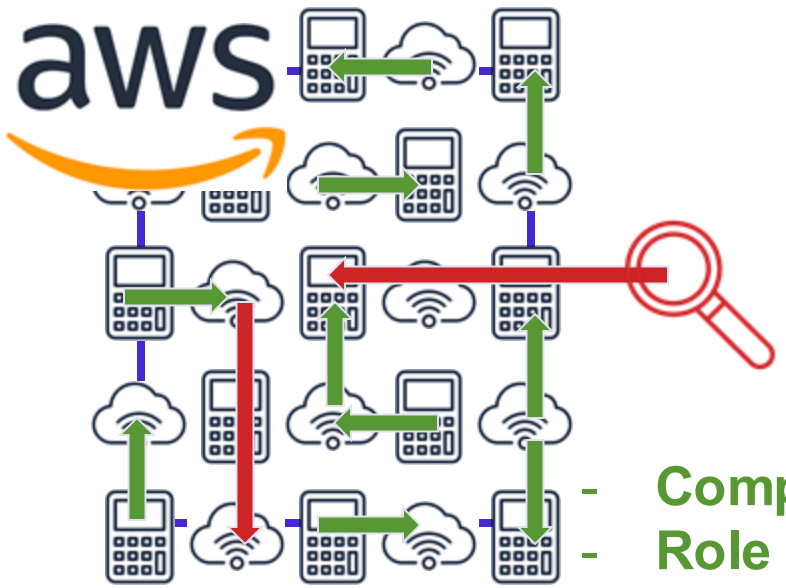
192.168.x.x/24

Exceeding service limits & Virtual Private Cloud (VPC) peering quotas



Governance is harder with larger accounts

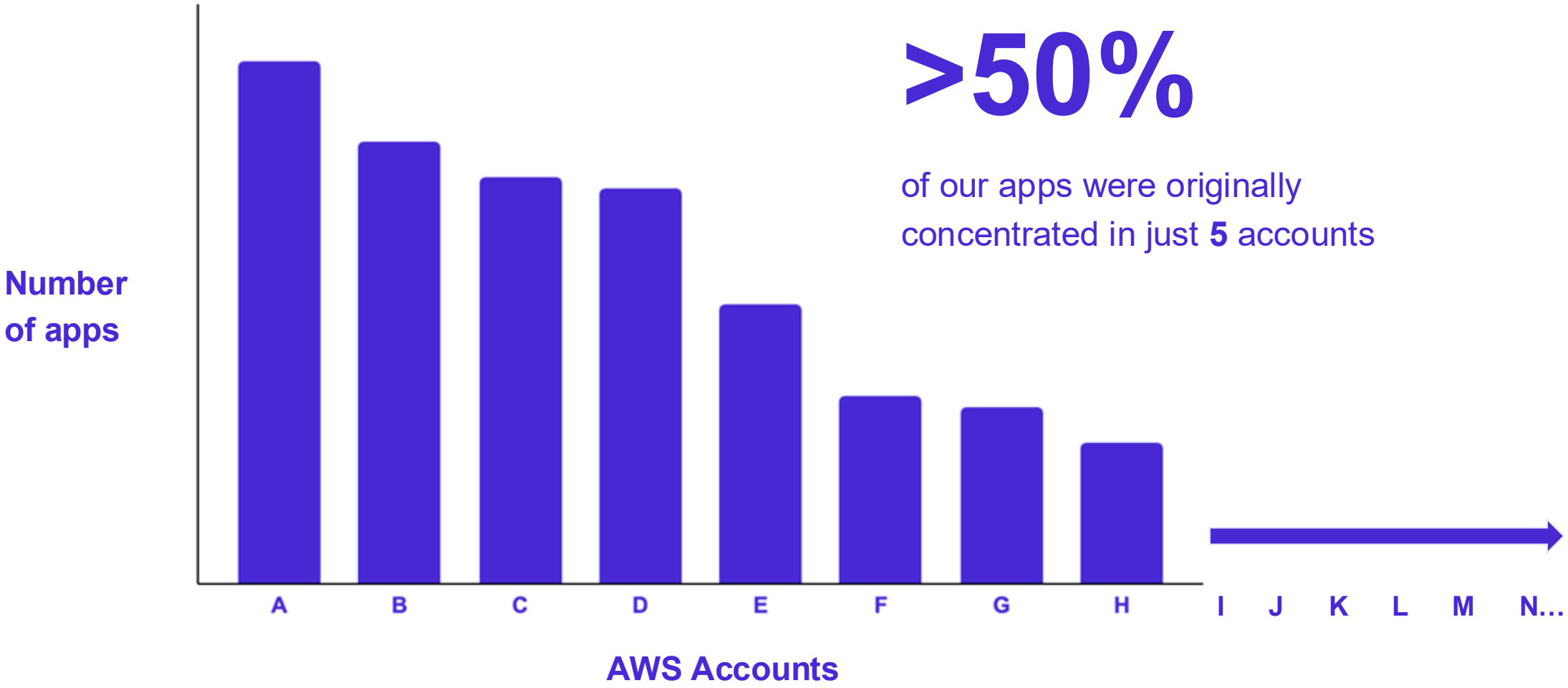
Complicated spend tracking per application



- Data perimeter & workload isolation
- Compliance with data-locality requirements

- Complicated routing rules
- Role shopping more likely

When we started this effort, Capital One had some very large AWS accounts



>50%

of our apps were originally concentrated in just **5** accounts

Source deck

Our large accounts were a legacy of our early migration to the cloud

2016

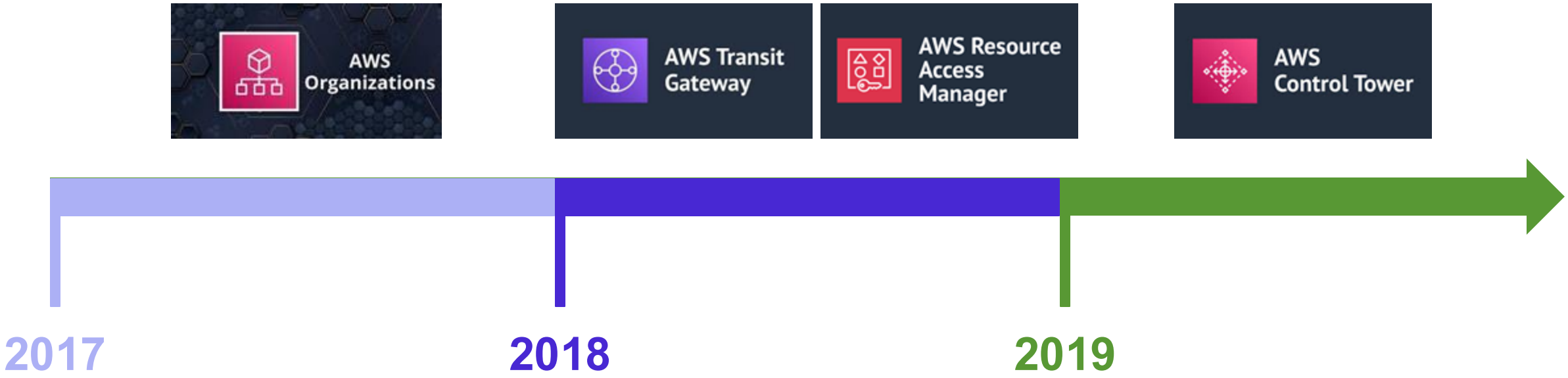
Capital One declares "all in" on the public cloud

but at the time…

- **AWS tools could not support many accounts**
- **Account structure guidance was sparse**
- **IAM service had fewer capabilities**

We closed our data centers in 2020

During our move to the cloud, AWS simultaneously released tools to enable a multi-account strategy



During our move to the cloud, AWS simultaneously released tools to enable a multi-account strategy

Policy-based management for multiple accounts

Centralized routing & segmentation policies

Enables resource sharing across multiple accounts

Rapidly provision and govern accounts

AWS' Well-Architected Framework now includes multi-account design principles from the start

- Organize based on security and operational needs
- Apply security controls to Organizational Units (OU) rather than accounts
- Avoid deep OU hierarchies
- Avoid deploying workloads to the organization's management account
- Separate prod from non-prod workloads
- Use a common identity provider to simplify managing human access to accounts

AND MORE…

AWS multi-account guidance preaches that smaller is better, but your organization decides on your structure

Org-level segmentation

X accounts per business unit

Y accounts per division

Workload-level segmentation

1 account per application

1 account per environment
E.g., Dev, Test, QA, Pre-prod, Prod

AWS limits:

10K accounts / organization

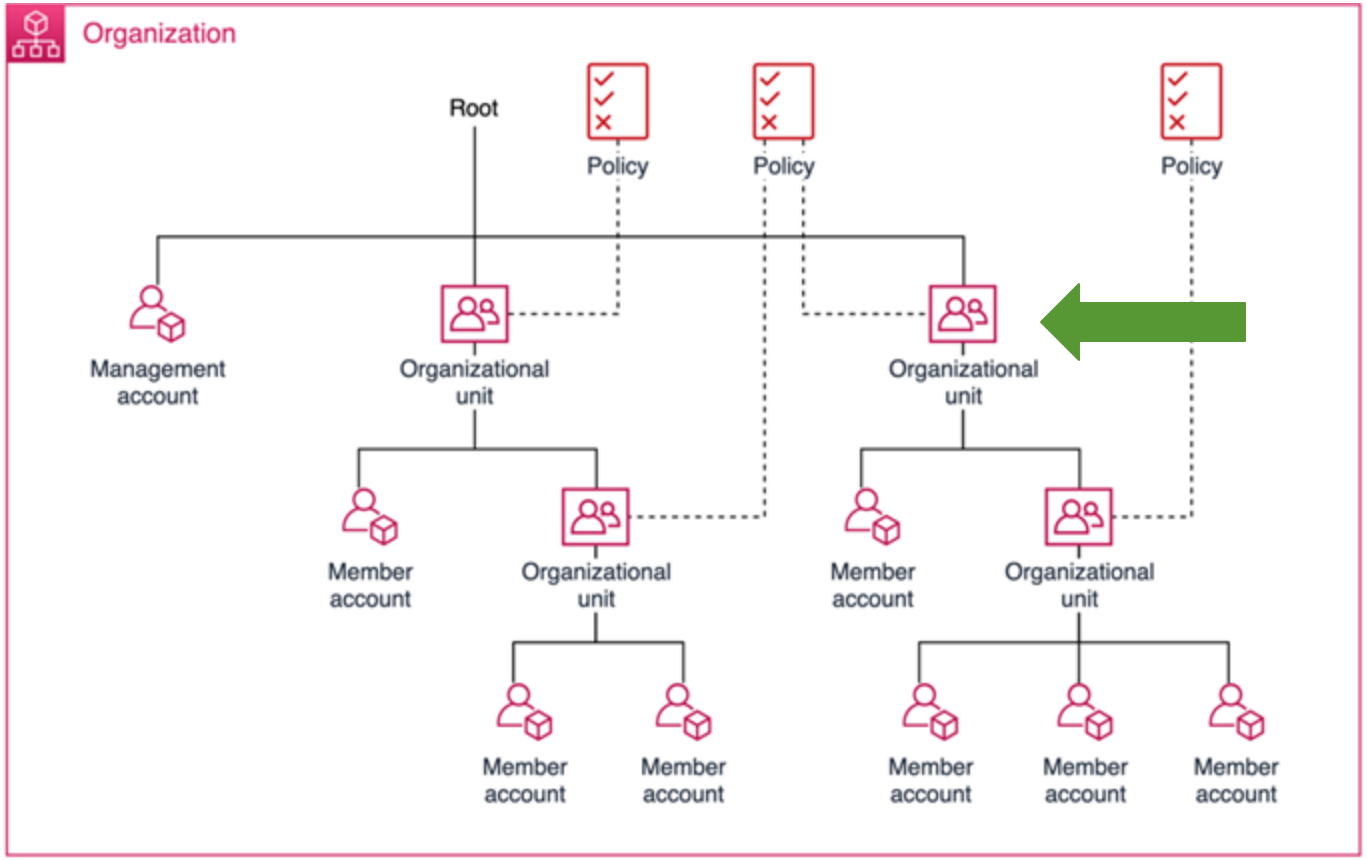
50 accounts / OU
(OUs can be nested)

300 groups
5K users

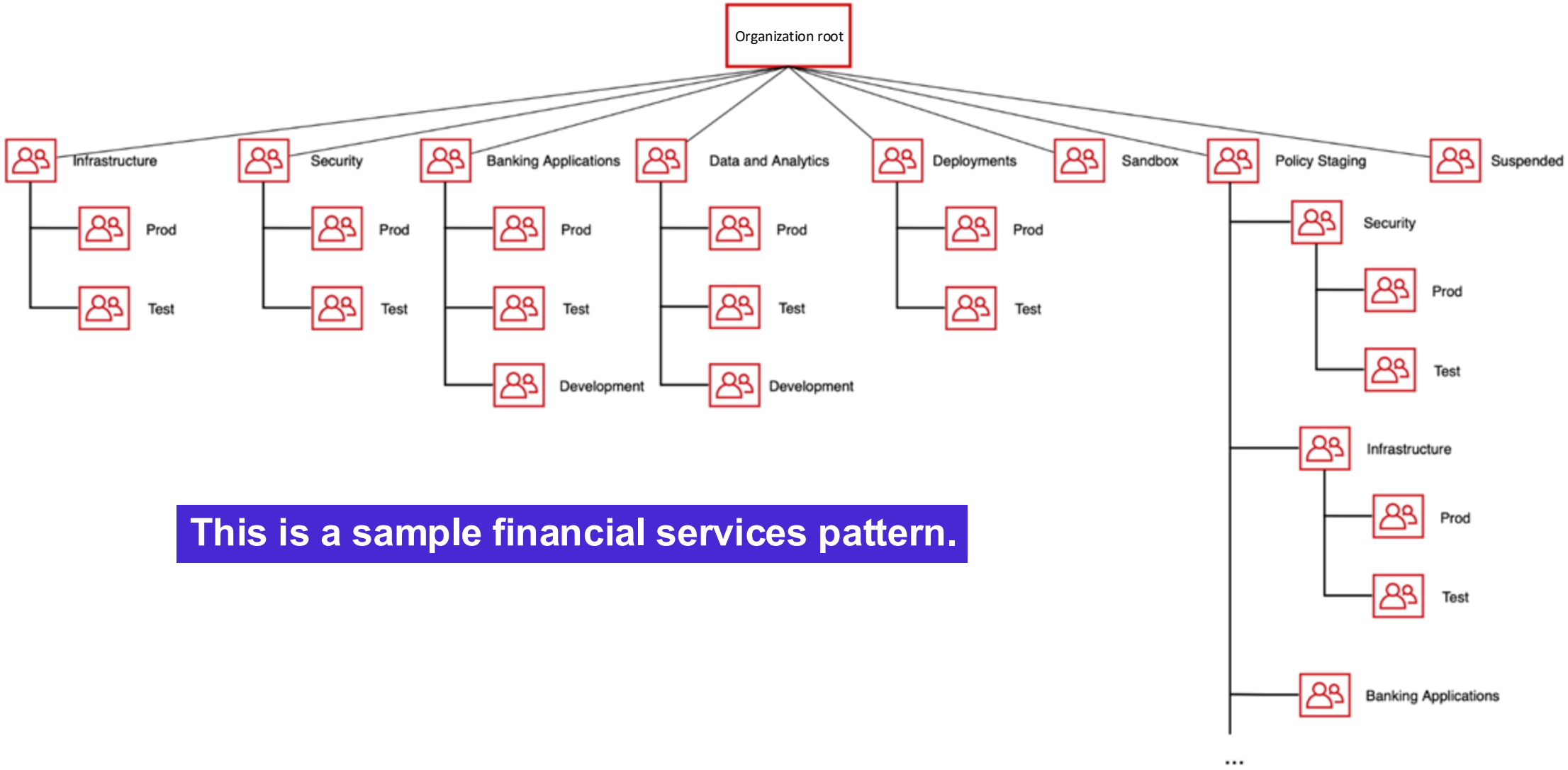
1.5K customer-managed policies

With a multi-account strategy, AWS Organizations are a must-have for central management and administration

- Centralized control of billing and account management
- Logical account hierarchy and segmentation aligned to your business functions
- Enterprise-level security policy enforcement through Service Control Policies (SCP) at the organization level



AWS provides sector-specific patterns for AWS Organizations



This is a sample financial services pattern.

Our AWS multi-account architecture (“Gen3”) segments network and application resources

Network accounts

Vends & hosts network resources

Controls the network boundary

E.g., VPC, subnet, route table, NACL, NAT Gateway, TGW, DXG, etc.

App accounts

Vends & hosts app resources

Controls the IAM boundary

E.g., EC2, RDS, S3, IAM roles, Security Group, etc.

To drive consistent security isolation, we established a tenancy strategy for the network versus IAM boundary

Network tenancy

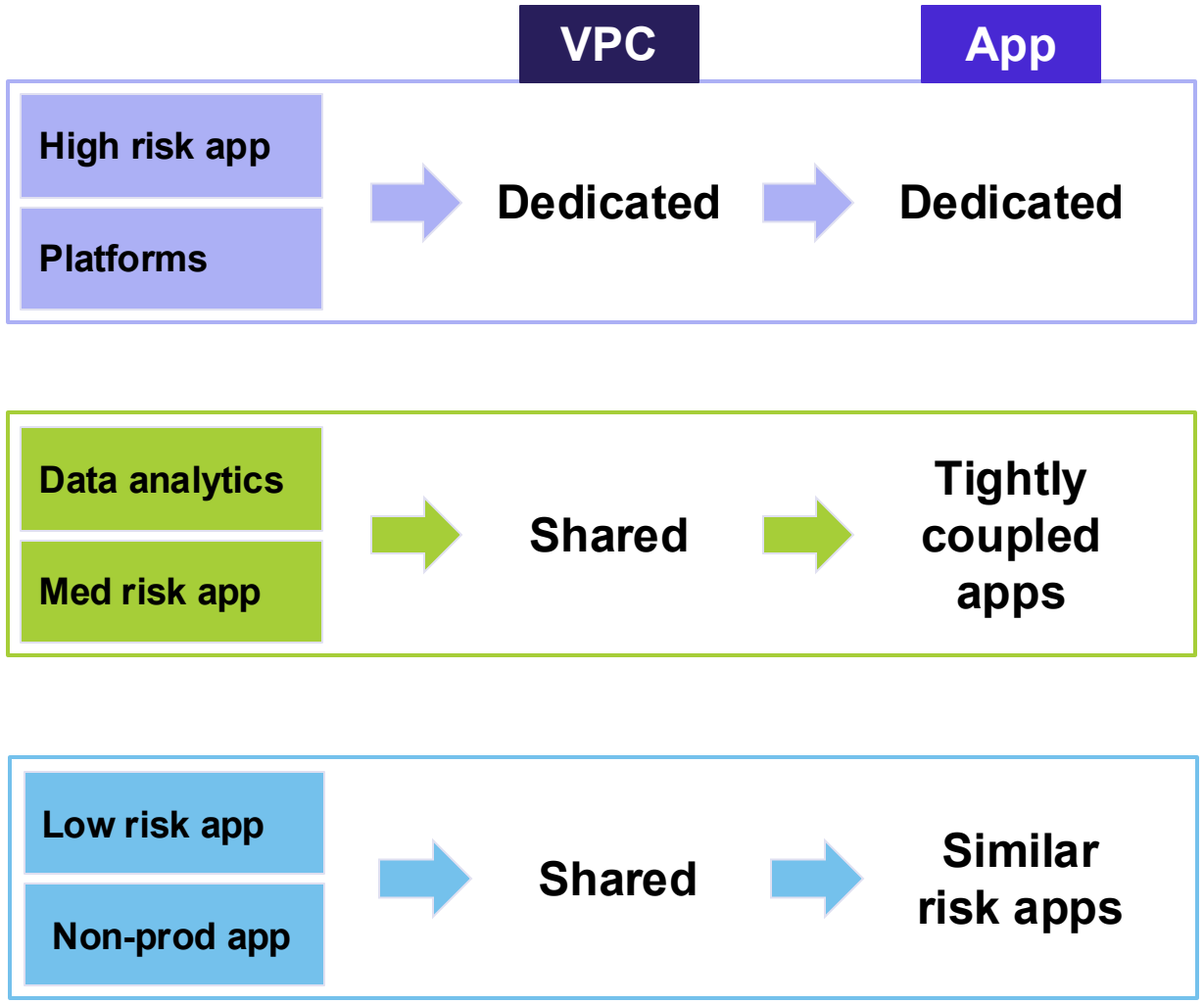
Dedicated VPC	Enables resources from a single app account to access and deploy resources within the VPC
Shared VPC	Enables resources from multiple app accounts to access and deploy resources within the VPC

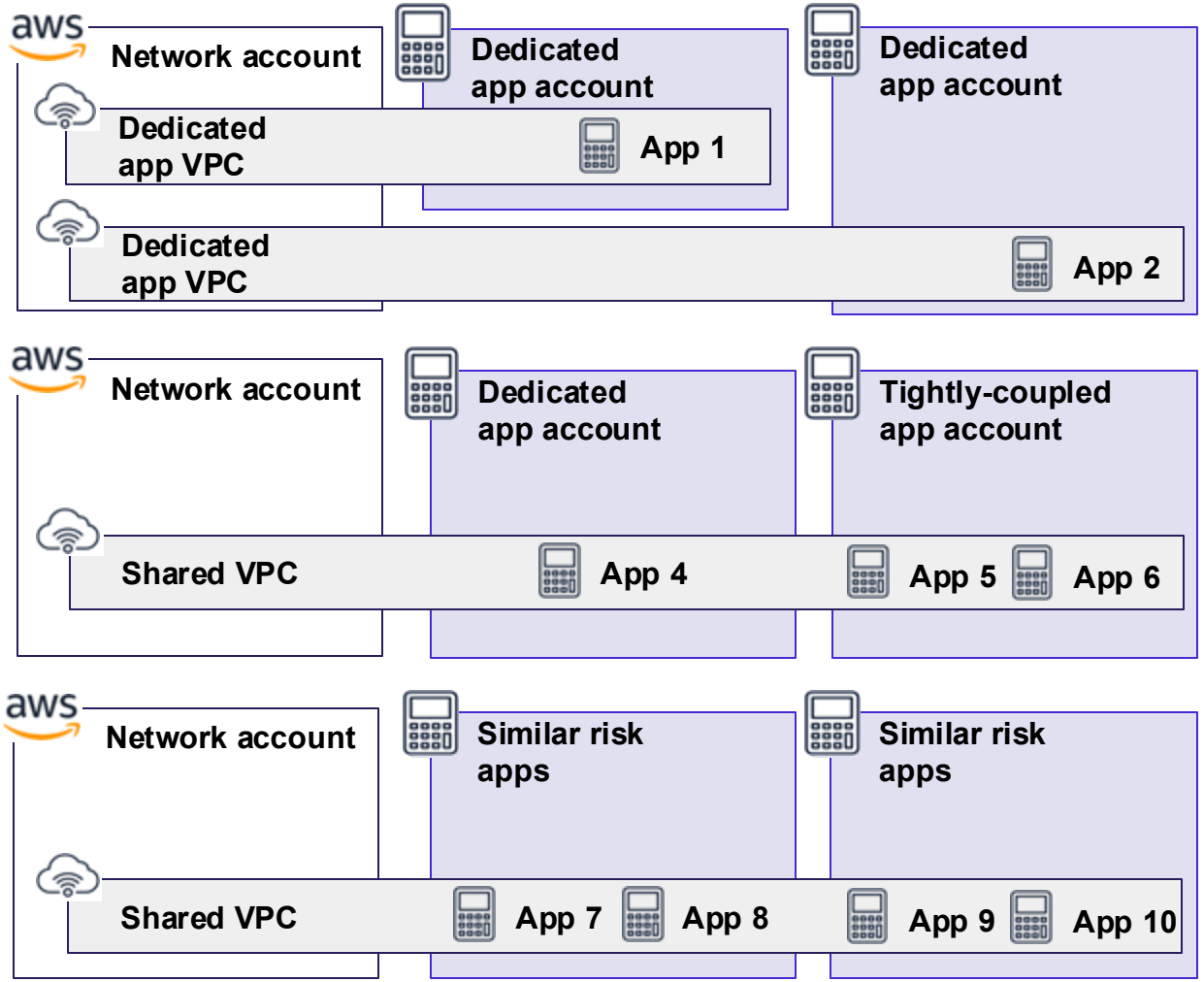
App tenancy

Dedicated app account	AWS account with resources supporting a single app (i.e., 1:1 app to account ratio)
Shared app account	AWS account with resources supporting multiple apps with common business and risk profile

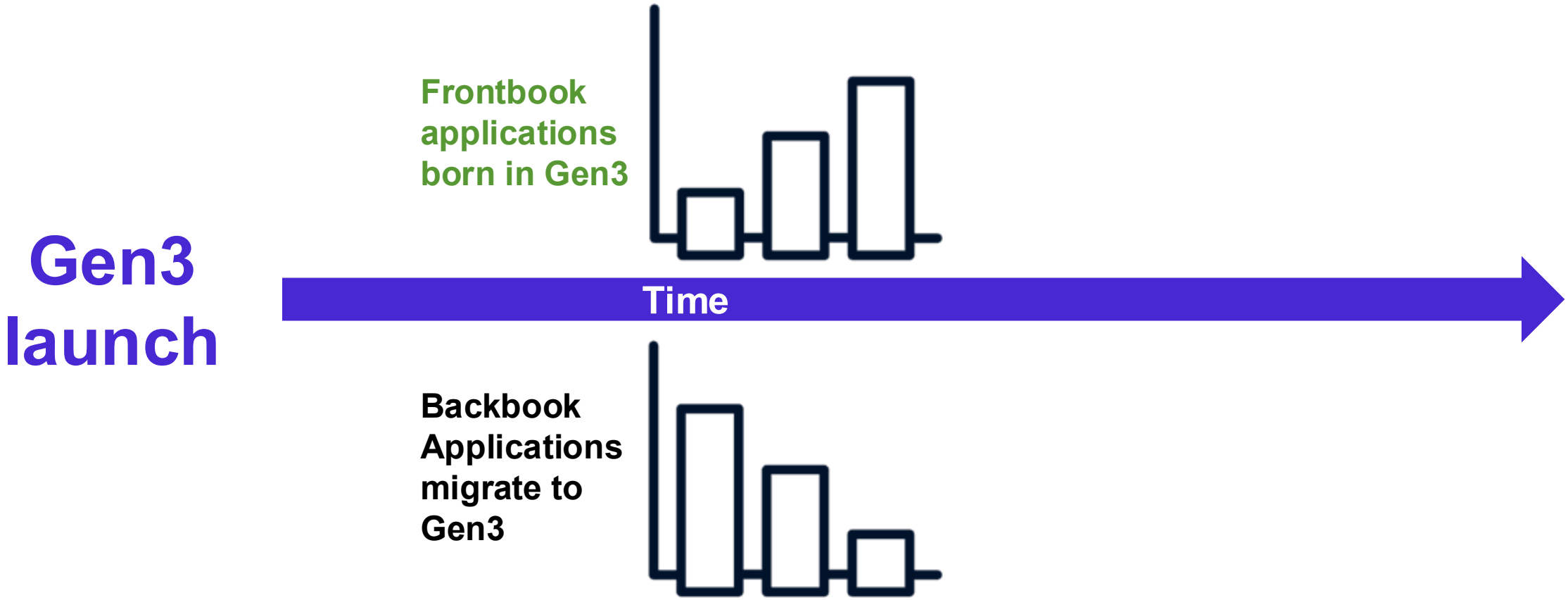
Network resources are always shared from network account to app accounts

We developed a decision tree to disposition apps into appropriate Gen3 patterns





To reduce the blast radius, we had to adopt AWS Gen3 accounts without disrupting our core business



To deliver AWS Gen3 accounts and VPCs at scale, we had to automate A LOT of the provisioning process

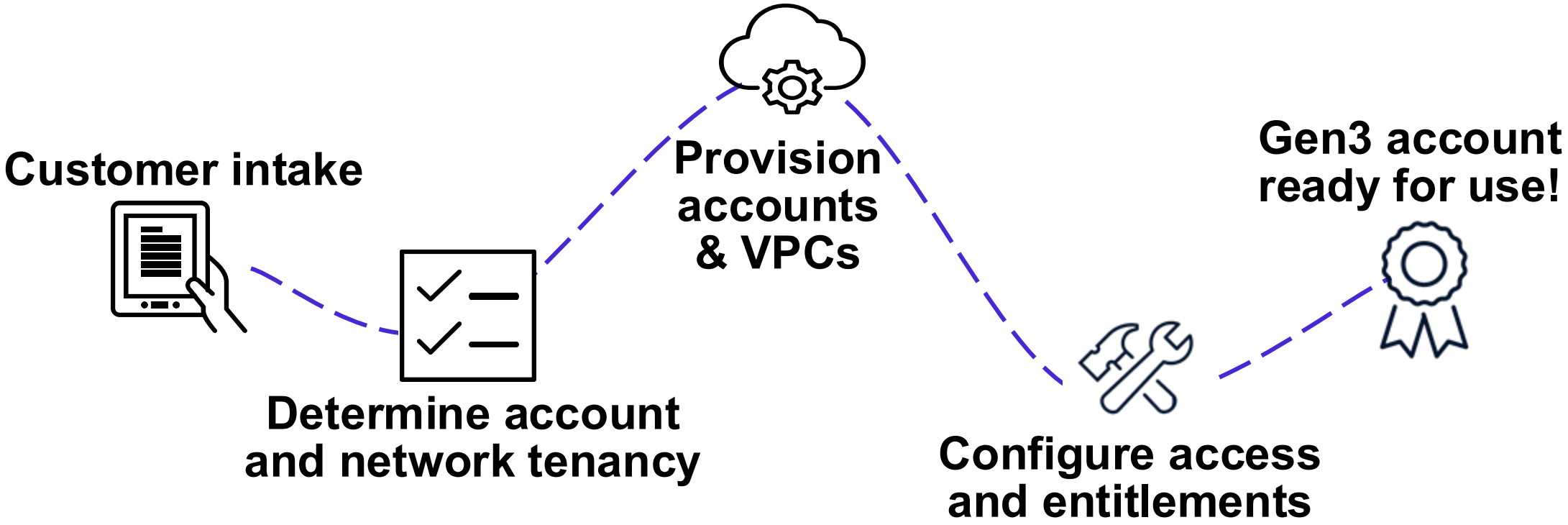
Network account setup

- DNS VPC:
  - Subnets
  - Network Access Control Lists (NACLs)
  - Private route tables
  - DNS security groups
  - Route53 resolver
- Network VPC:
  - Subnets
  - NACLs
  - Route table
  - Security groups
  - Parameter groups
  - Subnet groups
  - Hosted zones
  - Transite Gateway (TGW) attachments
  - Route53 Resolver association

Application account setup

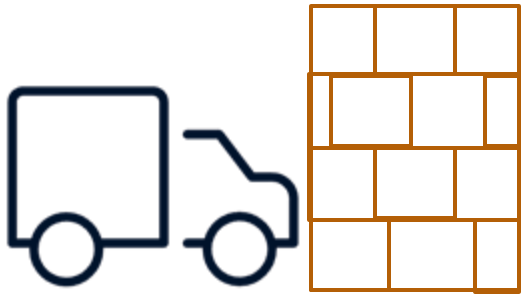
- Multiple accounts and VPCs per region and environment type
- Security groups
- Parameter and subnet groups
- Hosted zones
- Amazon Certificate Manager (ACM) certificates
- Key Management Service (KMS) keys & S3 bucket policies
- Removal of default network resources
- IAM role and access provisioning
- Resources shared from network account:
  - VPC DHCP
  - VPC flow logs
  - subnets
  - NACLs
  - route tables

We provision AWS Gen3 accounts through an intent-based service-enablement process


Customer intake
Determine account and network tenancy
Provision accounts & VPCs
Configure access and entitlements
Gen3 account ready for use!

We enforce compliance via design, build, and launch readiness gates in our pipeline

There were challenges in automating the AWS Gen3 provisioning process



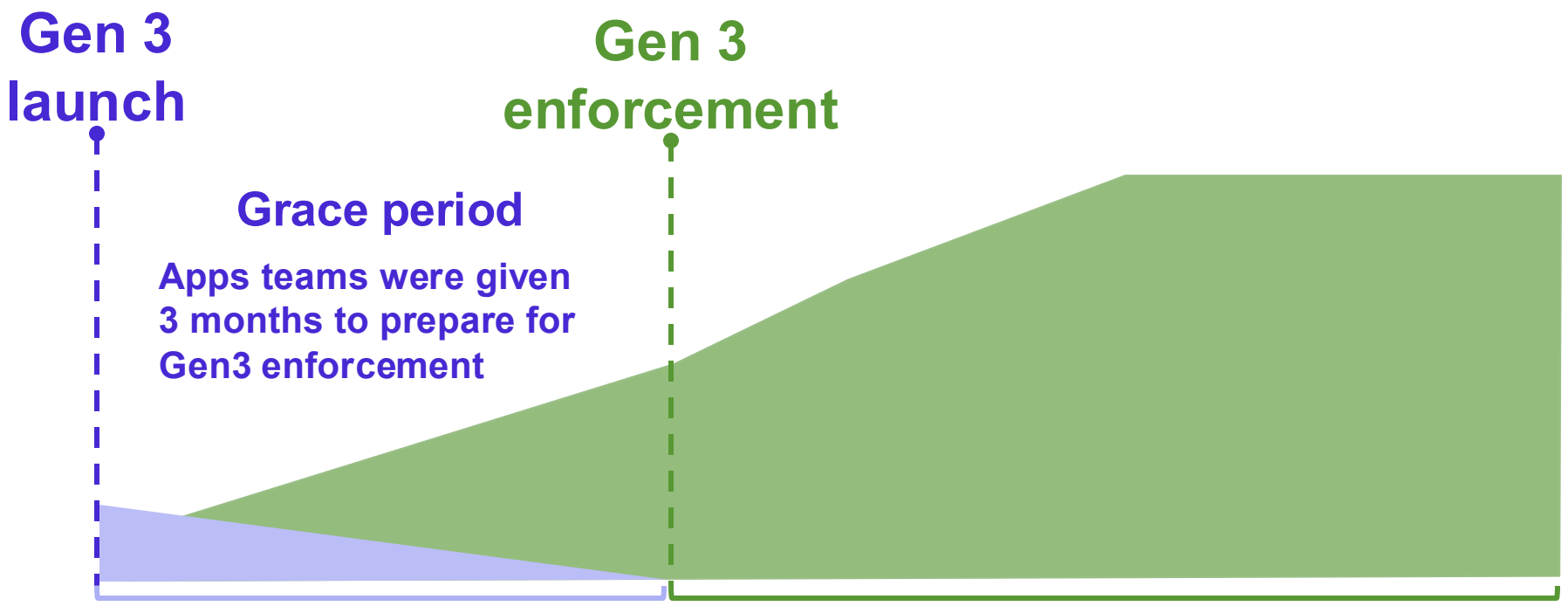
Development roadblocks

- Microservice failures cascaded with limited observability across teams
- Hand-offs were complex and had to be manually coded for different scenarios
- Diverse requirements
- AWS service quotas
- Manual internal program tracking
- Limited frontend customer onboarding support
- Lack of parallel account provisioning
- Manual validation and rollbacks

Cross team dependencies

- Vulnerability and Configuration Management
- IAM
  - IAM role deployment / AD groups
  - AuthN and AuthZ (IdP, SAML)
  - Key management policies
- Network Management
  - TGW Routing Management
  - IP Management
  - Developer pipeline enablement

After establishing the process and tools, we pushed frontbook apps to be born in AWS Gen3 accounts



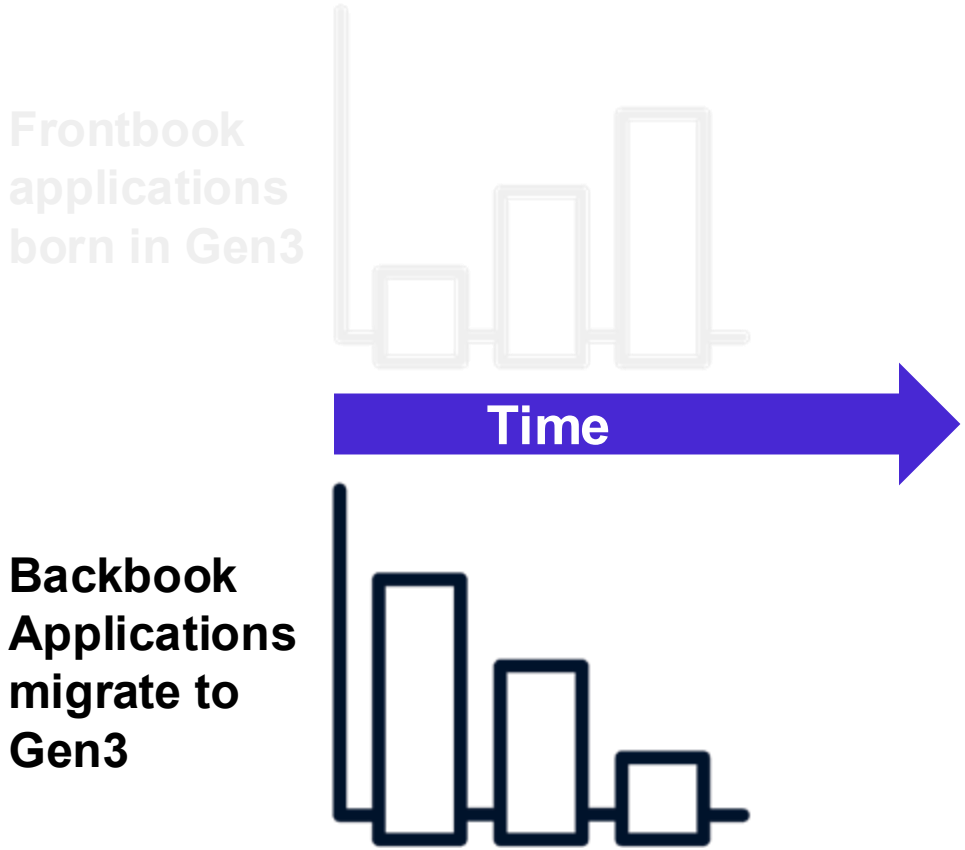
~99% of new apps built into Gen3 accounts

100% of new apps built into Gen3 accounts over last 2+ years

<1% Short-term exceptions for apps built using legacy accounts

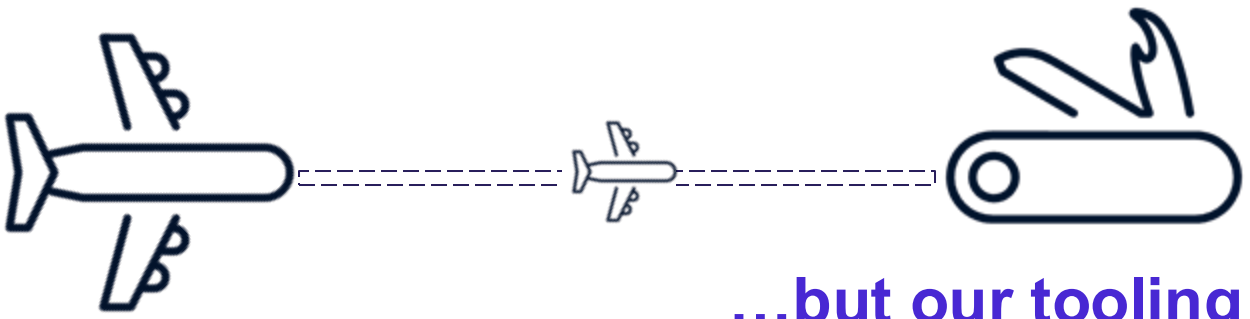
Backbook migration is much harder, and we're not done

GEN3 launch



From our initial Gen3 migration efforts, we learned that our tooling had to improve

We started with a pilot…



…but our tooling was not up to the task…

i.e., high provisioning & migration times for developers

…so, we invested in improving our tools

i.e., workflow orchestration as code to provision faster; cross account migration tooling improvements to migrate faster

After we built tooling, we built a risk-based approach to prioritize migrations

Risk-based prioritization

Adoption of new services

As AWS debuts new services, we enable and mandate them only in Gen3 accounts

Risk scoring factors

- Account size
- Resource counts / service limits
- PCI, SOX, Data Sensitivity

Infrastructure dependencies & complexity

Identify low-complexity apps based on AWS service usage and data volume, along with dependency maps for apps

Migration tool compatibility

Match migration tools' capability with services utilized by an app. Prioritize based on existing & evolving toolkit

We also take long-term data considerations into account to drive down our backbook population

Age out

App owners determine data that can be retired over time

E.g., S3 lifecycle policies, old (and unused!) data

Delete

Instead of migrating, old resources can be deleted and decommissioned

E.g., old backups and snapshots

Given the scale of the work, we had to motivate our engineers

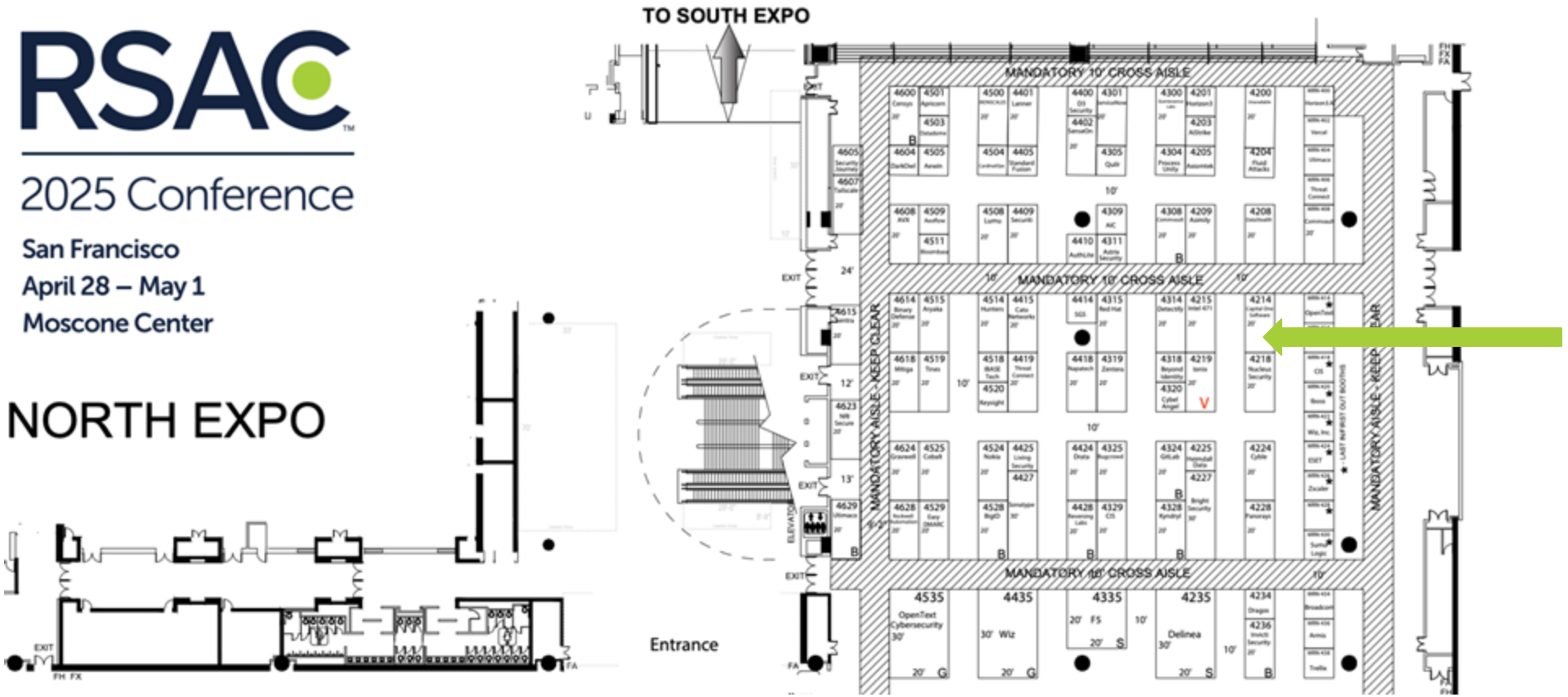
Reward	+	*Consequence*
Newly approved AWS services only available in Gen3 accounts **Opportunity to refactor**		**Made migration an enterprise priority** **Regular reporting on progress and escalation when deadlines missed**

Key takeaways for reducing your blast radius through AWS' multi-account strategy

Well-managed architecture	Provisioning improvements	Risk sloping and prioritized action	Migration tooling improvements
• Consider your business structure and tech stack to define your account structure	• Automate your management plane; *e.g., AWS Control Tower*	• Risk slope your backbook apps to drive migration planning	• Match your tools' capabilities with your apps' most commonly used AWS services
• Define logic that drives network and application account tenancy	• Use workflow-as-code to provision infrastructure	• Prioritize low complexity apps to rapidly reduce the overall blast radius	• Automate and integrate migration tools into the developer experience
• Stay true to your patterns; avoid bespoke deployments	• Measure time to provision; create self-service progress dashboards for devs	• Allow newly approved AWS services only in target account topology	• Build migration completion metrics into dashboards

Apply what you have learned today:

- Next week you should:
  - Identify AWS accounts with high concentrations of app and network resources
  - Determine the initial extent to which these accounts mix and/or share network and app resources
- In the first three months following this presentation you should:
  - Evaluate your AWS Orgs and account structure against AWS' Well-Architected principles
  - Develop common architecture patterns for AWS accounts and tenancy criteria for your apps' use cases
  - Identify your most common cloud services, profile your apps and their dependencies, and identify their target network and account tenancy pattern
  - Evaluate current cloud management processes, tooling, and automation; identify new services and tools needed to support the AWS account management lifecycle
  - Create an implementation roadmap
- Within six months you should:
  - Begin implementing, develop lessons learned, adapt, and repeat!

Q&A

Visit Capital One Software at the expo (N-4214):